Tutorial 10

Exercise 1

(a) Suppose the sequence of integers 23, 9, 17, 88, 3, 27, 30 is to be inserted into a binary search tree (BST). Then, the root of the BST contains the integer 23 after the first insertion. Draw the BST after inserting all the integers in that sequence.

(b) Suppose for every node x of an AVL tree, its balance factor is computed as follows:

balance factor of x = height of right subtree of x - height of left subtree of x

Figure 1 shows the AVL tree containing the integer 23 with the balance factor 0 after the first insertion. Based on Figure 1, draw the AVL tree and indicate the balance factors for all nodes after inserting the sequence of integers in part (a) of this question.

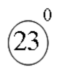

Figure 1

(c) Draw the binary search tree after deleting node 48 from the binary search tree in Figure 2.

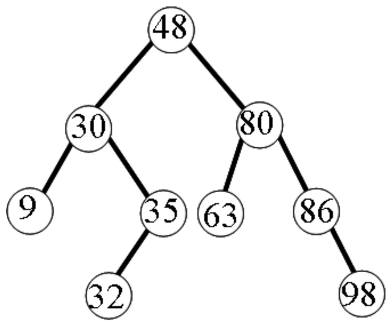

Figure 2

(d) If preorder traversal is used to visit the binary tree in Figure 2, show the order of the nodes visited.

Exercise 2

```
class Node:
   def __init__(self, key):
      self.left = None
      self.right = None
      self.key = key

   def insert(self, key):
    # Compare the new value with the parent node
      if self.key:
         if key < self.key:
            if self.left is None:
               self.left = Node(key)
            else:
               self.left.insert(key)
         elif key > self.key:
               if self.right is None:
                  self.right = Node(key)
               else:
                  self.right.insert(key)
      else:
         self.key = key


    # To find the inorder successor which is the smallest node in th
e subtree
   def findsuccessor(self, node):
        current_node = node
        while current_node.left != None:
            current_node = current_node.left
        return current_node

   def delete(self, root, key):
      # Base Case
      if root is None:
          return root

      # If the key to be deleted
      # is smaller than the root's
      # key then it lies in  left subtree
      if key < root.key:
          root.left = self.delete(root.left, key)

      # If the kye to be delete
```

```
        # is greater than the root's key
        # then it lies in right subtree
        elif(key > root.key):
            root.right = self.delete(root.right, key)

        # If key is same as root's key, then this is the node
        # to be deleted
        else:

            # Node with only one child or no child
            if root.left is None:
                temp = root.right
                root = None
                return temp

            elif root.right is None:
                temp = root.left
                root = None
                return temp

            # Node with two children:
            # Get the inorder successor
            # (smallest in the right subtree)
            temp = self.findsuccessor(root.right)

            # Copy the inorder successor's
            # content to this node
            root.key = temp.key

            # Delete the inorder successor
            root.right = self.delete(root.right, temp.key)

        return root

# Print the tree
   def PrintTree(self):
      if self.left:
         self.left.PrintTree()
      print(self.key),
      if self.right:
         self.right.PrintTree()

# Use the insert method to add nodes
root = Node(20)
root.insert(10)
root.insert(40)
root.insert(6)
root.insert(15)
root.insert(18)
root.insert(30)
root = root.delete(root, 15)
root.PrintTree()
```

a) Implement the BST, for figure 2.
b) Is the display showing Pre-order, In-order or Post-order?
c) Add a method to show the pre-order traversal.